AUTORESEARCHCLAW: SELF-REINFORCING AUTONOMOUS RESEARCH WITH HUMAN-AI COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Automating scientific discovery requires iterative research pipelines that rigorously generate hypotheses, recover from experimental failures, retain useful experience across research cycles, and maintain strict data provenance. Existing systems often struggle with these demands: hypothesis generation and evaluation are frequently conflated, failed experiments are repaired at the code level without revisiting the underlying design, and reported numerical claims are rarely verified against execution records. We present AUTORESEARCHCLAW (ARC), an open-source 23-stage multi-agent pipeline that turns a research question into executed experiments, per-hypothesis verdicts, and a manuscript. ARC combines structured multi-agent debate, self-healing execution with PIVOT/REFINE decisions, registry-based result verification, configurable human intervention, and cross-run evolution that converts failures into reusable lessons. We evaluate ARC on ARC-BENCH, a 55-topic benchmark spanning machine learning, high-energy physics, metabolic modeling, statistics, and quantum computing. On 25 ML topics, ARC scores 0.596, compared with 0.511 for AIDE and 0.419 for AI Scientist v2. Cross-run self-reinforcement further raises overall performance from a mean fresh-run baseline of 0.564 to 0.827 after three persistent iterations, with the largest gains in execution and result analysis. End-to-end studies show that targeted intervention at high-leverage stages outperforms both full autonomy and step-by-step oversight, while component ablations show that debate improves quality, self-healing improves completion, and the measurement registry keeps unsupported numbers out of the gate-passing manuscripts audited in our ablation.

1 INTRODUCTION

Automating scientific discovery is a long-standing goal of artificial intelligence. Recent LLM-based systems can propose hypotheses, write and run experiment code, and draft papers (Lu et al., 2024; Yamada et al., 2025; Schmidgall et al., 2025; Tang et al., 2025). Real research, however, does not proceed in a straight line from idea to paper: a researcher proposes a hypothesis, designs an experiment, observes what fails, revises the plan, and tries again. This process depends on four capabilities: challenging one's own hypotheses from several angles, recovering from failed experiments without abandoning the line of work, retaining useful lessons across research cycles, and reporting only what was actually measured.

Existing systems handle these capabilities unevenly. Systems such as the AI Scientist (Lu et al., 2024) generate and assess research directions within the same pipeline, making weak assumptions and overly easy directions difficult to surface. Tree-search agents such as AIDE (Jiang et al., 2025) and AI Scientist v2 (Yamada et al., 2025) repair failing code inside a search tree, but experimental failure does not directly return the system to hypothesis generation. Numerical claims in generated write-ups are also seldom checked against execution records: MLR-Bench reports that coding agents frequently produce fabricated or invalid experimental results (Chen et al., 2025a). A further question is whether experience from one completed research cycle can improve subsequent cycles rather than being discarded after each run.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

These challenges interact: better hypotheses reduce downstream revisions, failure-aware execution preserves the measurements that analysis depends on, grounded analysis turns failures into useful evidence, and retained experience allows those lessons to inform later research cycles. We therefore address them together in a single framework.

We present AUTORESEARCHCLAW (ARC), an open-source 23-stage research pipeline (Figure 1) built around five mechanisms. *Structured multi-agent debate* assigns three roles (*e.g.* Innovator, Pragmatist, Contrarian) that critique each other at hypothesis generation and again at result analysis, with a synthesizer integrating their assessments. *Self-healing execution* diagnoses and repairs failing experiments, and a PROCEED/REFINE/PIVOT decision after result analysis either keeps the direction or returns a flawed design to hypothesis generation. *Verifiable result reporting* drafts tables from a registry of executed measurements, replaces any unmatched number in results sections and tables with a placeholder, and verifies every citation through a four-layer pipeline. *Human-in-the-loop collaboration* provides intervention modes from full autonomy to step-by-step approval, at which a researcher can approve, edit, or add guidance. *Cross-run evolution* persists lessons extracted from failures and research decisions across runs, ranks them by recency and relevance, and injects them into subsequent stage prompts.

We evaluate ARC on ARC-BENCH, a benchmark of 55 topics in machine learning, high-energy physics, metabolic modeling, statistics, and quantum computing. Our contributions are as follows.

- ARC, an open-source multi-agent research pipeline that unifies debate, self-healing execution with a hypothesis-level PIVOT, registry-based number checking, human-in-the-loop modes, and cross-run evolution, released with ML and HEP-ph prompt banks, adapters for eight further fields, and domain agents for HEP-ph, metabolic modeling, and statistics.
- ARC-BENCH, with hierarchical rubrics that adapt the requirement structure of PaperBench (Starace et al., 2025) to comparative studies without a reference paper, and an experiment-stage protocol that grades systems with different end-to-end capabilities on the same outputs: code, logs, and write-up.
- On the 25 ML topics, ARC in Full-Auto mode scores 0.596 against 0.511 for AIDE and 0.419 for AI Scientist v2, with the largest gains on result analysis. On the HEP-ph topics, ARC's domain agent successfully reproduces published cross-section curves.
- Cross-run evolution improves mean Overall performance from 0.564 across three reset runs to 0.670 across three independent memory-enabled Iteration-1 runs; continuing the original persistent sequence reaches 0.827 at Iteration 3. End-to-end studies further characterize human intervention and the distinct failure modes addressed by debate, self-healing, and the measurement registry.

2 RELATED WORK

Autonomous research systems. LLM-based systems increasingly automate substantial parts of the scientific workflow. In chemistry, autonomous agents have planned and executed laboratory experiments (Boiko et al., 2023); in algorithm discovery, program-search systems have produced new mathematical and algorithmic results (Romera-Paredes et al., 2024; Novikov et al., 2025).

The AI Scientist (Lu et al., 2024) automates idea generation, code writing, experiment execution, visualization, manuscript generation, and simulated review. AI Scientist v2 (Yamada et al., 2025) removes the dependence on human-written experiment templates and introduces progressive agentic tree search inspired by AIDE (Jiang et al., 2025), which formulates ML experimentation as tree search over executable code candidates. Agent Laboratory (Schmidgall et al., 2025) organizes research into literature review, experimentation, and report writing, while allowing human feedback at each stage. AI-Researcher (Tang et al., 2025) similarly spans literature review, hypothesis generation, algorithm implementation, and publication-ready manuscript preparation.

Other systems emphasize different parts of the research loop. Baby-AIGS (Liu et al., 2024) introduces explicit falsification: experimental results are used to formulate candidate explanations and test them through further experiments and ablations. This form of scientific hypothesis verification is distinct from checking whether numerical claims in a generated manuscript agree with the underlying execution records. AgentRxiv (Schmidgall & Moor, 2025) provides a shared preprint server through which autonomous agent laboratories can retrieve and build on prior agent-generated reports. Co-Scientist (Gottweis et al., 2026) focuses on hypothesis generation and research planning, using

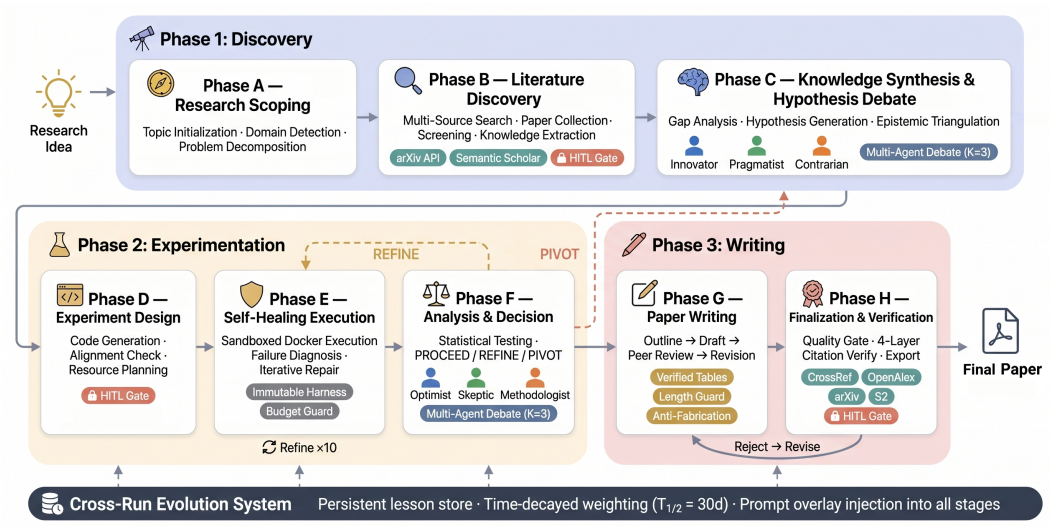

Figure 1: Overview of the AUTORESEARCHCLAW pipeline. Twenty-three stages span **Discovery**, **Experimentation**, and **Writing**: structured debate supports hypothesis generation and result analysis, failure-aware execution enables iterative repair and REFINE/PIVOT decisions, and verified measurements ground manuscript drafting. HITL gates enable targeted intervention, while the cross-run lesson store carries experience into subsequent research cycles. Detailed stage definitions and control flow are given in the Appendix A.

specialized agents, scientific debate, and tournament-style ranking; its reported wet-lab validations are subsequently performed by scientists rather than autonomously by the agent.

Table 1 summarizes mechanisms explicitly described by representative research agent systems. ARC combines hypothesis-level feedback, programmatic result verification, citation verification, human intervention, and non-ML scientific toolchains within a single pipeline.

Evaluating research agents. Existing benchmarks isolate different parts of research automation. MLAgentBench (Huang et al., 2024) and MLE-bench (Chan et al., 2025) evaluate iterative ML experimentation and engineering using executable performance metrics. ScienceAgentBench (Chen et al., 2025b) derives data-driven scientific tasks from peer-reviewed papers and evaluates generated programs, their execution results, and associated costs. DiscoveryBench (Majumder et al., 2024) formalizes multi-step data-driven hypothesis discovery, while DiscoveryWorld (Jansen et al., 2024) evaluates complete discovery cycles in a simulated environment with automatic metrics for task completion, scientific actions, and discovered explanatory knowledge.

ARC-BENCH complements these settings by evaluating end-to-end research improvement across five domains using task-specific executable metrics and rubric-based comparative judgments, without requiring a single reference paper as the target output.

Debate, experience, and human involvement. Multi-agent debate has been studied as a mechanism for improving factuality and reasoning (Du et al., 2024; Liang et al., 2024), while role-structured systems such as ChatDev (Qian et al., 2024), MetaGPT (Hong et al., 2024), and AutoGen (Wu et al., 2024) organize collaboration among specialized agents. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use feedback to revise behavior within a task, whereas SkillRL (Xia et al., 2026) and EvolveR (Wu et al., 2026) distill past trajectories into reusable skills or strategic principles that can guide later tasks.

Human guidance remains important in research systems. Agent Laboratory reports that human feedback improves overall research quality (Schmidgall et al., 2025). SciSciGPT (Shao et al., 2026) and AIssistant (Gaddipati et al., 2026) explicitly study human–AI collaborative research workflows, and Natarajan et al. (2025) argue that evaluation should treat the human as an active component of the overall system rather than scoring the AI component in isolation. Our human-in-the-loop ablation contributes empirical evidence to this question: with the content of the guidance held fixed

Table 1: Capabilities explicitly described by representative research systems. ✓ supported; ∼ partial or indirect support; ✗ outside the stated system scope; – not specified clearly enough to score. *Hypothesis-level pivot* means that failed experimental evidence can return the pipeline to hypothesis generation. *Result verification* means that reported numerical claims are programmatically checked against execution records.

Capability	AIDE	AI Sci. v2	Agent Lab	AI Res.	Baby-AIGS	Co-Sci.	ARC
Autonomous experiment execution	✓	✓	✓	✓	✓	✗	✓
Code-level self-repair	✓	✓	✓	–	–	–	✓
Hypothesis-level pivot	✗	∼	✗	–	∼	✗	✓
Multi-agent debate	✗	✗	∼	–	–	✓	✓
Persistent research memory	✗	–	–	–	–	∼	✓
Result verification	✗	–	–	–	–	✗	✓
Citation verification	✗	∼	–	–	–	–	✓
Human-in-the-loop mode	✗	✗	✓	–	–	✓	✓
Manuscript generation	✗	✓	✓	✓	–	✗	✓
Non-ML scientific toolchains	✗	✗	✗	✗	✗	∼	✓
Open source	✓	✓	✓	✓	✓	–	✓

across seven intervention regimes, targeted intervention at high-leverage stages outperforms both full autonomy and exhaustive step-by-step oversight.

3 AUTORESEARCHCLAW

ARC runs 23 stages in the eight phases of Figure 1, from Research Scoping to Finalization & Verification; Appendix A lists the stages and gives the control flow. Each stage declares its required input and output artifacts, a completion criterion (*e.g.* at least two falsifiable hypotheses), and an error code, and a run can resume from any completed stage. A domain profile selects a prompt bank and an adapter that supplies domain terminology, standard baselines, and libraries; for HEP-ph, metabolic modeling, and statistics, it also routes execution to a domain agent equipped with the field's toolchain (Appendix B). We call the pipeline *self-reinforcing* because failures feed back at two timescales: within a run, iterative repair and REFINE/PIVOT revise the current experiment or hypotheses; across runs, retained lessons from earlier research cycles are injected into subsequent stage prompts.

3.1 MULTI-AGENT DEBATE

A single LLM agent tends to confirm the hypotheses it generates, because the model that proposes an idea has no structural incentive to disconfirm it. ARC therefore runs a structured debate ($K=3$; Appendix H) at hypothesis generation (stage 8) and again at result analysis (stage 14). Three role-conditioned agents write their assessments independently, and a synthesizer integrates them and records the disagreements that remain. At stage 8, an *Innovator* proposes high-risk hypotheses, a *Pragmatist* checks them against the hardware and time budget, and a *Contrarian* looks for confounds and trivial explanations; the synthesizer produces two to four falsifiable hypotheses with measurable predictions, failure conditions, and required baselines. At stage 14, after statistical testing of the measurements, an *Optimist*, a *Skeptic*, and a *Methodologist* assess the results, and the synthesizer assigns each hypothesis a verdict of supported, refuted, or inconclusive before any writing begins. Domain-specific prompt banks replace these roles with field-specific counterparts; the complete role mapping is given in Appendix B.

3.2 SELF-HEALING EXECUTION AND RESEARCH DECISIONS

Experiment failure is common in real research, and a system that terminates on the first failure avoids ambitious experiments by design; ARC instead treats failure as diagnostic information. A code agent writes a per-file blueprint and generates files in dependency order using AST summaries of the files already written, and an *alignment check* regenerates code that does not test the stated topic. Before any execution budget is spent, static checks send code back for regeneration when two ablation conditions share identical code or when metric values are hard-coded, and a short pilot

run warns when all conditions produce identical metrics. Generated code is screened for forbidden calls and modules and runs in a sandbox (stage 12; a subprocess sandbox by default, or Docker with a staged network policy); metrics are reported through an *immutable harness* (`report_metric`, `finalize`), and a *budget guard* enforces wall-clock limits (Appendix C). *Iterative repair* (stage 13) reruns the experiment for up to ten iterations, revising the code from the previous output; *failure diagnosis* adds a targeted repair when validation fails or metrics are NaN, and the best version is kept. After result analysis, the decision stage (15) selects one of three actions: PROCEED when the evidence supports the hypotheses; REFINE, which keeps the hypotheses and returns to iterative repair when results are weak but the direction is sound; or PIVOT, which returns to hypothesis generation (stage 8) with the failure recorded as evidence. REFINE and PIVOT share a budget of two decisions, after which the pipeline proceeds.

3.3 VERIFIABLE RESULT REPORTING

LLM-generated papers often feature fabricated metrics and hallucinated citations—producing fluent, authoritative-sounding text that lacks any real evidentiary support. ARC addresses both with deterministic checks. During execution, every value produced by experiment runs, including per-condition means, standard deviations, and per-seed measurements, is recorded in a *measurement registry*. The drafting stage (17) does not start until a usable metric has been recorded; it then builds *verified tables* from the registry and instructs the writing prompt to copy them and to report no number absent from the experiment data. The prompt can read the registry but not modify it. As a final *anti-fabrication* check before export (stage 22), a verifier extracts the numbers in results sections and tables and matches each against the registry within a 1% relative tolerance, including rounded and percentage forms; unmatched numbers are replaced with a visible placeholder, and numbers in other sections are logged as warnings. *Four-layer citation verification* resolves each reference through CrossRef (DOI), OpenAlex (title), arXiv (identifier), and Semantic Scholar (title), before drafting and again at stage 23; unresolved entries and entries that an LLM judges irrelevant are removed together with their in-text citations, unless this would remove more than half of the bibliography.

3.4 HUMAN-IN-THE-LOOP COLLABORATION

Full automation loses quality at junctures where domain judgment matters, and approval at every step forfeits the efficiency of automation. ARC provides intervention modes that let researchers choose an operating point between these extremes. *Full-Auto* runs without pauses. *Gate-Only* pauses for approval at three HITL gates: literature screening (stage 5), experiment design (stage 9), and the quality gate (stage 20). *Thorough* pauses at the eight phase boundaries. *CoPilot* concentrates interaction at high-leverage decisions: collaborative sessions at synthesis, hypothesis generation, experiment design, and drafting (stages 7, 8, 9, 17), approval at stages 5, 8, 9, 15, and 20, and output review at eight further stages (Table 16). *Step-by-Step* pauses after every stage. At each pause the researcher can approve the stage outputs, reject them to stop the run, edit an output file, or open a multi-turn chat with an LLM that rewrites the stage's artifacts; written guidance is saved and added to the prompts of the later stages that read guidance (9, 14, 16, 17, and 21).

3.5 SELF-REINFORCING MECHANISM: CROSS-RUN EVOLUTION

Stateless systems repeat failures that earlier runs already encountered. When a run ends, ARC's evolution system appends lessons to a persistent *lesson store*, extracted from failed or blocked stages, REFINE and PIVOT rationales, and runtime anomalies such as warnings and NaN metrics. Each lesson records its stage, one of six categories, a severity level, and a description. When a run reads the store, the lessons for each stage are ranked by

$$\omega(l) \;=\; b(l) \cdot \exp\!\big(-\ln 2 \cdot \Delta t / T_{1/2}\big)\,, \tag{1}$$

where Δt is the age of lesson l, $T_{1/2}$=30 days is a half-life, and the boost factor $b(l)$ is 2 for lessons recorded at the stage being prompted, 1.5 for error-level lessons, and their product when both apply. The lesson store persists across consecutive research runs, so each iteration can reuse experience accumulated by the previous one; an optional skill bridge exports selected lessons beyond the current project.

Table 2: Results on ML01–ML25. Δ: mean paired difference (in percentage points).

System	CD	CE	RA	Overall	Δ	ARC higher
ARC	0.938	**0.562**	**0.442**	**0.596**	–	–
AIDE	**0.958**	0.415	0.336	0.511	+8.5%	18/25
AI Scientist v2	0.712	0.442	0.261	0.419	+17.7%	21/25

4 EXPERIMENTS

We evaluate ARC through five studies: a controlled comparison with existing research agents on the 25 ML topics of ARC-BENCH together with cross-domain coverage (§4.2); a study of cross-run self-reinforcement (§4.3); an end-to-end evaluation under seven human-in-the-loop regimes (§4.4); a component ablation (§4.5); and a case study (§4.6).

4.1 SETUP

Benchmark. ARC-BENCH specifies 55 topics: 25 CPU-executable ML topics (ML01–ML25) on classical learning tasks, from tabular learning and optimization to causal discovery and learning to rank; 10 high-energy-physics reproductions with MadGraph5_aMC@NLO (Alwall et al., 2014) (P01–P10); 7 constraint-based metabolic modeling studies with COBRApy (Ebrahim et al., 2013) and BiGG models (King et al., 2016) (B01–B07); 3 statistics studies such as bootstrap coverage (Efron, 1979) and double machine learning (Chernozhukov et al., 2018) (S01–S03); and 10 quantum computing studies, mostly with variational circuits (Q01–Q10). Each topic names a research question, the conditions to compare, a primary metric, and two to four hypotheses (Appendix D).

Rubric and judge. Each topic is graded with a hierarchical rubric spanning code development (CD), code execution (CE), and result analysis (RA), weighted 25:25:50 in the Overall score. Two independent agent judges score de-identified run directories, and disagreements are re-adjudicated. The strict evidence protocol, leaf-level criteria, judge prompts, and sensitivity to alternative category weights are given in Appendix D.2 and Appendix E.

Studies and baselines. In the experiment-stage protocol, every system receives the manifest's hypotheses and experiment plan; the ML comparison therefore isolates code generation, execution, and result analysis (ARC stages 10–14). The HITL ablation and the component ablation run all 23 stages and score the resulting manuscripts from 1 to 10 with an LLM reviewer that applies a fixed rubric in two averaged passes (the pipeline's stage-20 quality gate), the same reviewer for every regime and configuration; human reviewers not involved in developing ARC additionally apply the same rubric to every completed manuscript and record pass or fail (Appendix F). We compare with AIDE (Jiang et al., 2025) and AI Scientist v2 (Yamada et al., 2025), whose execution paths can be driven by the same manifest, using their released code; all three systems use GPT-5.3-codex for every LLM role and the same total wall-clock budget of 1,800 s per run. Agent Laboratory (Schmidgall et al., 2025), which starts from literature review, did not reach its experimentation phase within our budget. Paired differences are tested two-sided with the Wilcoxon signed-rank test (topic-level scores) or a paired t-test (manuscript scores).

4.2 MAIN RESULTS: COMPARISON WITH RESEARCH AGENTS

ARC outperforms both baselines. In Full-Auto mode, ARC scores 0.596 overall, compared with 0.511 for AIDE and 0.419 for AI Scientist v2 (Table 2), corresponding to relative improvements of 17% and 42%, respectively. ARC scores higher than AI Scientist v2 on 21 of 25 topics, with a mean paired advantage of 17.7 percentage points (95% bootstrap interval: 9.9–25.6 points), and higher than AIDE on 18 of 25 topics, with an 8.5-point advantage (1.1–15.6 points; Figure 3 in Appendix E). The ranking is unchanged under three alternative aggregations of the same leaf scores and on the 15 topics not used during development, where ARC maintains advantages of 9.2 percentage points over AIDE and 17.8 points over AI Scientist v2 (Appendix E).

Result analysis and execution separate the systems. ARC's largest consistent advantage is in result analysis: it exceeds AIDE on RA on 19 of 25 topics by 10.6 points on average, while its CE advantage

Table 3: Persistent cross-run self-reinforcement on ML01–ML25. Three independent fresh runs estimate the reset no-memory reference. The trajectory shown on the right follows one memory-enabled sequence for three iterations, retaining lessons between consecutive cycles. Complete replication is reported in Appendix 12. $\Delta_{\text{Base}\to 3}$ denotes the difference between Iteration 3 and the mean fresh-run reference.

	Fresh-Run Baseline (No Memory)				Self-Reinforcement Trajectory			
Metric	**Run 1**	**Run 2**	**Run 3**	**Base Avg**	**Iter. 1**	**Iter. 2**	**Iter. 3**	$\boldsymbol{\Delta_{\text{Base}\to 3}}$
CD	0.938	0.844	0.912	0.898	0.948	0.968	**0.976**	+0.078
CE	0.562	0.426	0.534	0.507	0.574	0.588	**0.738**	+0.231
RA	0.442	0.358	0.476	0.425	0.612	0.703	**0.796**	+0.371
Overall	**0.596**	**0.497**	**0.600**	**0.564**	**0.687**	**0.741**	**0.827**	**+0.263**

is larger in magnitude but less consistent. Six AIDE runs exhaust the shared 1,800-s budget, whereas ARC's repair loop allows failed experiments to be revised within the same budget. The Skeptic and Methodologist roles target the complementary RA failure mode by challenging whether each hypothesis verdict is supported by the corresponding measurements.

Coverage beyond machine learning. The 30 non-ML topics require domain toolchains for collider simulation, metabolic modeling, statistical inference, and quantum computing. ARC routes them to tool-equipped domain agents (the HEP-ph agent is adapted from ColliderAgent (Qiu et al., 2026)); across all Non-ML topics, AIDE averages 0.23 and produces no write-up on 16 tasks, whereas ARC in CoPilot mode scores 0.82 on average (Appendix D.3).

4.3 Cross-Run Self Reinforcement

We evaluate cross-run self-reinforcement in two complementary ways. First, three fresh runs with the lesson store reset provide a no-memory reference, while three independent memory-enabled Iteration-1 runs test whether the initial cross-run gain is reproducible across independent executions. Second, we continue the original memory-enabled sequence for two further iterations, retaining lessons between cycles, to examine how performance evolves as experience accumulates. Table 3 shows the persistent trajectory, while the complete independent Iteration-1 replication is reported in Appendix 12. All runs use the same CD:CE:RA weighting as §4.

Performance improves with retained cross-run experience. The three fresh no-memory runs average 0.564 Overall. At the first memory-enabled iteration, three independent runs score 0.687, 0.654, and 0.670, for a mean of 0.670—an improvement of 0.106 over the fresh-run reference.(Appendix 12).

Continuing the original persistent sequence further increases Overall from 0.687 at Iteration 1 to 0.741 at Iteration 2 and 0.827 at Iteration 3. Relative to the fresh-run mean, the Iteration 3 differences are +0.078 in CD, +0.231 in CE, and +0.371 in RA. The independent Iteration-1 runs show that the initial improvement is not confined to a single execution, while the persistent trajectory shows how performance evolves as lessons accumulate across additional research cycles.

4.4 Human-in-the-Loop Ablation

Design. To evaluate the full pipeline, we run all 23 stages on ML01–ML10 under seven intervention regimes (the five modes of §3.4 and two decompositions of Step-by-Step defined below), one run per topic and regime, and score each manuscript with the stage-20 quality gate. To make individual interventions reproducible across regimes, the guidance payloads are fixed in advance: for each topic, one written payload was prepared for each of stages 5, 8, 9, 14, 17, and 20 (60 payloads, drafted by an LLM from the topic manifest and preliminary runs and edited by the authors), and a scripted adapter delivers the payload when a regime pauses at its stage and approves the stage output otherwise. *Pre-Experiment* receives only the payloads for stages 5, 8, and 9 and *Post-Experiment* only those for stages 14, 17, and 20, both under the Step-by-Step schedule (Table 16).

Targeted intervention beats both extremes. CoPilot obtains the highest score under both the automated gate and human review despite receiving fewer intervention opportunities than Step-

Table 4: HITL ablation on ML01–ML10. *Payloads*: number delivered (the stage-20 payload arrives after scoring). *Compl.*: runs with a scored manuscript. *Score*: mean stage-20 score over completed runs, and over all topics with incomplete runs scored as 1. *Pass, gate*: score ≥ 5, over all ten topics. *Pass, human*: mean number of manuscripts passed by three independent reviewers under the same rubric, over all ten topics.

Regime	Pauses	Payloads	Compl.	Score (compl.)	Score (all)	Pass, gate	Pass, human
Full-Auto	0	0	8/10	4.03	3.42	2/10	2.3/10
Gate-Only	3	3	10/10	5.03	5.03	5/10	4.0/10
CoPilot	15	5	8/10	**7.27**	**6.02**	**7/10**	**6.3/10**
Thorough	8	1	7/10	4.86	3.70	3/10	3.3/10
Step-by-Step	23	6	10/10	5.19	5.19	5/10	5.7/10
Pre-Experiment	23	3	8/10	4.28	3.62	3/10	3.7/10
Post-Experiment	23	3	6/10	5.08	3.45	3/10	3.3/10

by-Step: it scores 3.21 points above Full-Auto on matched topics (p=0.019), 2.16 points above Step-by-Step, and passes 7 of 10 topics (Table 4). The human reviewers pass 6.3 CoPilot manuscripts on average compared with 2.3 for Full-Auto, with Step-by-Step second at 5.7.

Intervention must span both sides of execution. Pre-Experiment guidance improves design feasibility but cannot repair later claim formation; Post-Experiment guidance improves completed manuscripts but cannot recover evidence absent from failed experiments. Gate-Only provides a lower-interaction alternative, completing all ten runs with three pauses.

4.5 Component Ablation

To isolate the contribution of each mechanism, we run Full-Auto on ML01–ML10 with one mechanism removed at a time, and with debate and self-healing removed together (Table 5); because runs are stochastic and path-dependent, each configuration–topic pair is run three times and the best-scoring completed run is kept.

Table 5: Component ablation. *Score*: mean stage-20 score over completed topics, and over all topics with incomplete ones scored as 1. *Pass, gate*: score ≥ 5, over all ten topics. *Pass, human*: mean number of manuscripts passed by independent reviewers under the same rubric (Appendix F); for the registry row the reviewers reported the unsupported numbers and did not record a pass decision. *Unsupp.*: gate-passing manuscripts that contain a number absent from all measurement records, over gate-passing manuscripts (human audit).

Configuration	Compl.	Score (compl.)	Score (all)	Pass, gate	Pass, human	Unsupp.
Full ARC	**10/10**	**5.62**	**5.62**	3/10	**2.3/10**	0/3
without debate	10/10	4.25	4.25	1/10	0.7/10	0/1
without self-healing	6/10	4.83	3.30	1/10	0.0/10	0/1
without registry	10/10	5.48	5.48	**5/10**	–	3/5
without debate & self-healing	4/10	3.47	1.99	0/10	0.3/10	–

Each mechanism addresses a distinct failure mode. Debate is the largest contributor to the quality of completed manuscripts: removing it lowers the mean score by 1.37 points, because without the Pragmatist filtering infeasible hypotheses and the Skeptic scrutinizing weak findings, both the hypotheses and the analysis degrade. Self-healing is the largest contributor to completion: without it only 6 of 10 topics complete even with three attempts each, because the first unrecovered runtime error ends the run, and the mean score with incomplete topics scored as 1 falls by 2.32 points. Removing both leaves 4 of 10 topics complete and no manuscript passes the automated quality gate:: debate without self-healing produces ambitious hypotheses that crash on the first failure, and self-healing without debate repairs experiments that test poorly formulated questions. Under the human check, the full system passes 2.3 of 10 manuscripts on average and every removal at most 0.7.

The registry is the integrity backstop. Without the registry, 5 manuscripts pass the quality gate, but human reviewers find unsupported numbers in 3 of them; with the registry, none of the 3 passing

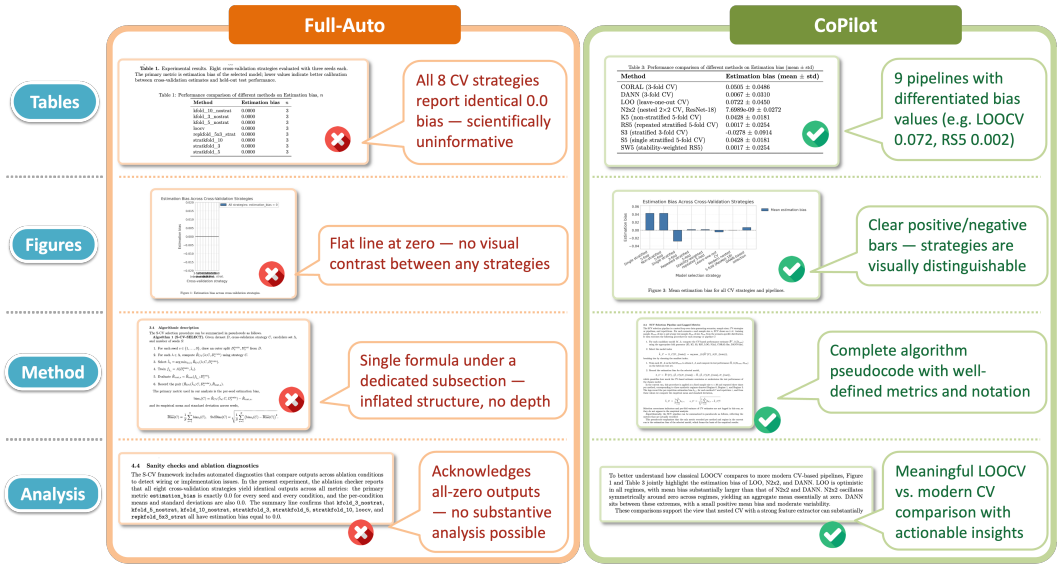

Figure 2: Case study comparing Full-Auto and CoPilot on Task ML10. Full-Auto suffers a silent semantic collapse: all cross-validation strategies report identical zero-bias outputs. CoPilot produces differentiated results, enabling a meaningful comparison across strategies.

manuscripts contains such numbers. The LLM quality gate alone does not catch these errors, while registry-grounded drafting and verification prevent them at the cost of a stricter automated pass rate.

4.6 Case Study: Cross-Validation Strategies (ML10)

We demonstrate the case study of ML10 task in Figure 2. ML10 compares cross-validation strategies for small-sample model selection; we contrast the Full-Auto and CoPilot runs of §4.4 (Appendix I). Both complete a manuscript, but Full-Auto fails in a way that runtime metrics do not capture: every strategy reports an estimation bias of exactly zero, the paper cannot support any comparison, and the gate scores it 4.0; the pilot-run check flagged the identical conditions, and the warning alone did not change the design. In this run, CoPilot avoids this failure; its stage-9 payload explicitly targets the experimental bottleneck by asking whether the strategies produce distinguishable outcomes, whether leave-one-out fits the time budget, and that the manuscript report only logged metrics.; the resulting paper reports differentiated bias values, states its limitations, and scores 8.0 (Appendix I). The case illustrates two points. The number check is necessary but not sufficient: Full-Auto passes it because its zeros are real measurements, and the check cannot tell whether those measurements answer the research question. The value of an intervention depends on where it is placed: guidance about experimental semantics before execution produced evidence-grounded output, and guidance at the writing stage could not have recovered evidence the experiment never produced.

5 Conclusion

We presented AutoResearchClaw, a multi-agent autonomous research pipeline combining structured debate, failure-aware execution, verifiable result reporting, human-in-the-loop collaboration, and cross-run evolution. Under a controlled experiment-stage protocol on 25 ML topics, ARC scores 0.596 compared with 0.511 for AIDE and 0.419 for AI Scientist v2. Cross-run self-reinforcement directly supports the system's central design: from a mean fresh-run baseline of 0.564, Overall performance reaches 0.827 after three persistent iterations, with the largest improvements in execution and result analysis. End-to-end studies further show that targeted intervention outperforms both full autonomy and step-by-step oversight, while ablations identify complementary roles for debate, self-healing, and registry-based numerical verification. Together, these results position ARC as a research system that can accumulate useful experience across research cycles while keeping experimental evidence and human scientific judgment explicit in the loop.

Ethics Statement

Systems that generate research papers can introduce fabricated results or non-existent references into the literature and can increase the number of low-quality submissions. ARC replaces numbers in results sections and tables that do not match executed measurements with a placeholder, and removes unresolved or irrelevant references unless doing so would eliminate more than half of the bibliography, in which case automatic deletion is suppressed. These checks establish that reported numbers were measured and references were checked, not that a scientific conclusion is correct. In our ablation without the registry, the LLM quality gate passed three manuscripts containing numbers absent from the measurement records; LLM review alone did not catch them, and it should not be the only integrity check. Researchers who use such systems remain responsible for the claims they submit and should disclose the use of autonomous research tools. Generated code is statically screened and runs under time limits; deployments that execute untrusted generated code should use the Docker backend with network isolation explicitly enabled and verified. The subprocess backend used in our experiments is intended for controlled research environments.

Reproducibility Statement

§3 and Appendices A–C describe the pipeline, prompts, and sandbox. §4.1 and Appendix D describe the topics, rubrics, judge protocol, baseline configurations, backbones, and budgets, and Appendix F lists the pause stages and payloads of each regime. The supplementary material will contain an anonymized repository with the system, the benchmark (topics, rubrics, and judge prompts), the 60 payloads, per-cell judge outputs, and stage-20 quality reports, with API credentials removed. Results depend on hosted LLMs whose behavior may change over time; we report the model used in every experiment.

AI Use Statement

LLMs are a component of the system studied in this paper (§3), act as judges in our evaluation (§4.1), and drafted the payloads of the HITL ablation, which the authors reviewed and edited (§4.4). In preparing the manuscript, the authors used LLM-based assistants for language editing. No part of the manuscript was generated by AutoResearchClaw. The authors checked all numbers against their experiment records and take full responsibility for the content of the paper.

References

J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7):79, 2014. doi: 10.1007/JHEP07(2014)079. URL https://link.springer.com/article/10.1007/JHEP07(2014)079.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023. doi: 10.1038/s41586-023-06792-0. URL https://www.nature.com/articles/s41586-023-06792-0.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Mądry, and Lilian Weng. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7e3767db483c942b883eb4f8cfb74e31-Abstract-Conference.html.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. Mlr-bench: Evaluating ai agents on open-ended machine learning research, 2025a. URL https://arxiv.org/abs/2505.19955.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench:

Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/pdf?id=6z4YKr0GK6.

Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018. doi: 10.1111/ectj.12097. URL https://academic.oup.com/ectj/article/21/1/C1/5056401.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Ali Ebrahim, Joshua A. Lerman, Bernhard O. Palsson, and Daniel R. Hyduke. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Systems Biology*, 7:74, 2013. doi: 10.1186/1752-0509-7-74. URL https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-7-74.

Bradley Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. doi: 10.1214/aos/1176344552. URL https://projecteuclid.org/journals/annals-of-statistics/volume-7/issue-1/Bootstrap-Methods-Another-Look-at-the-Jackknife/10.1214/aos/1176344552.full.

Sasi Kiran Gaddipati, Farhana Keya, Gollam Rabby, and Sören Auer. AIssistant: Human-AI Collaborative Review and Perspective Research Workflows in Data Science. In *Advances in Knowledge Discovery and Data Mining: 30th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2026, Proceedings*, volume 16600 of *Lecture Notes in Computer Science*, pp. 499–511. Springer, 2026. doi: 10.1007/978-981-92-1468-6_35. URL https://link.springer.com/book/10.1007/978-981-92-1468-6?page=2.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist, 2026. URL https://arxiv.org/abs/2502.18864.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20271–20309. PMLR, 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. DISCOVERYWORLD: A Virtual Environment for Developing and Evaluating Automated Scientific Discovery Agents, 2024. URL https://arxiv.org/abs/2406.06769.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-Driven Exploration in the Space of Code, 2025. URL https://arxiv.org/abs/2502.13138.

Zachary A. King, Justin Lu, Andreas Dräger, Philip Miller, Stephen Federowicz, Joshua A. Lerman, Ali Ebrahim, Bernhard O. Palsson, and Nathan E. Lewis. BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research*, 44(D1):D515–D522, 2016. doi: 10.1093/nar/gkv1049. URL https://academic.oup.com/nar/article/44/D1/D515/2503057.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate, 2024. URL https://arxiv.org/abs/2305.19118.

Zijun Liu, Kaiming Liu, Yiqi Zhu, Xuanyu Lei, Zonghan Yang, Zhenhe Zhang, Peng Li, and Yang Liu. AIGS: Generating Science from AI-Powered Automated Falsification, 2024. URL https://arxiv.org/abs/2411.11910.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2019. URL https://papers.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. DiscoveryBench: Towards Data-Driven Discovery with Large Language Models, 2024. URL https://arxiv.org/abs/2407.01725.

Sriraam Natarajan, Saurabh Mathur, Sahil Sidheekh, Wolfgang Stammer, and Kristian Kersting. Human-in-the-loop or AI-in-the-loop? Automate or Collaborate? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28594–28600, 2025. doi: 10.1609/aaai.v39i27.35083. URL https://ojs.aaai.org/index.php/AAAI/article/view/35083.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Shi Qiu, Zeyu Cai, Jiashen Wei, Zeyu Li, Yixuan Yin, Qing-Hong Cao, Chang Liu, Ming xing Luo, Xing-Bo Yuan, and Hua Xing Zhu. An End-to-end Architecture for Collider Physics and Beyond, 2026. URL https://arxiv.org/abs/2603.14553.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Samuel Schmidgall and Michael Moor. AgentRxiv: Towards Collaborative Autonomous Research, 2025. URL https://arxiv.org/abs/2503.18102.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM Agents as Research Assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.320. URL https://aclanthology.org/2025.findings-emnlp.320/.

Erzhuo Shao, Yifang Wang, Yifan Qian, Zhenyu Pan, Han Liu, and Dashun Wang. SciSciGPT: advancing human–AI collaboration in the science of science. *Nature Computational Science*, 6:301–315, 2026. doi: 10.1038/s43588-025-00906-6. URL https://www.nature.com/articles/s43588-025-00906-6.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://papers.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research, 2025. URL https://arxiv.org/abs/2504.01848.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous Scientific Innovation. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0320. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/0d904d300a105809a2114d727851e759-Abstract-Conference.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen llm applications via multi-agent conversation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-Evolving LLM Agents through an Experience-Driven Lifecycle, 2026. URL https://arxiv.org/abs/2510.16079.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning, 2026. URL https://arxiv.org/abs/2602.08234.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, 2025. URL https://arxiv.org/abs/2504.08066.

Table 6: Complete 23-stage pipeline specification, grouped by the phases of Figure 1.

#	Stage Name	Key Output	Special
Phase A: Research Scoping			
1	TOPIC_INIT	SMART goal, hardware profile	Domain detection
2	PROBLEM_DECOMPOSE	Problem tree	
Phase B: Literature Discovery			
3	SEARCH_STRATEGY	Search queries, `sources.json`	
4	LITERATURE_COLLECT	Candidate papers	Multi-source search
5	LITERATURE_SCREEN	Shortlisted papers	HITL gate
6	KNOWLEDGE_EXTRACT	Knowledge cards	
Phase C: Knowledge Synthesis & Hypothesis Debate			
7	SYNTHESIS	Gap analysis	
8	HYPOTHESIS_GEN	2–4 hypotheses	Debate (K=3), epistemic triangulation
Phase D: Experiment Design			
9	EXPERIMENT_DESIGN	`exp_plan.yaml`	HITL gate
10	CODE_GENERATION	Multi-file experiment code	Code agent, alignment check
11	RESOURCE_PLANNING	Time/resource estimates	
Phase E: Self-Healing Execution			
12	EXPERIMENT_RUN	Results	Sandbox, immutable harness, budget guard
13	ITERATIVE_REFINE	Improved code + metrics	Iterative repair ($\leq$10 iterations)
Phase F: Analysis & Decision			
14	RESULT_ANALYSIS	Analysis + charts	Statistical testing, debate (K=3)
15	RESEARCH_DECISION	Proceed / Refine / Pivot	Decision
Phase G: Paper Writing			
16	PAPER_OUTLINE	Section outline	
17	PAPER_DRAFT	Full paper draft	Verified tables, anti-fabrication
18	PEER_REVIEW	Review feedback	Three simulated reviewers
19	PAPER_REVISION	Revised paper	Length guard
Phase H: Finalization & Verification			
20	QUALITY_GATE	`quality_report.json`	HITL gate
21	KNOWLEDGE_ARCHIVE	Retrospective archive	
22	EXPORT_PUBLISH	PDF + sanitized BibTeX	Number check (anti-fabrication)
23	CITATION_VERIFY	`verification_report.json`	Four-layer citation verification

A FULL STAGE DEFINITIONS

Table 6 lists all 23 pipeline stages and Algorithm 1 gives the control flow. Each stage declares a *contract*: its required input artifacts, its expected output artifacts, a human-readable definition of done (*e.g.* "at least two falsifiable hypotheses"), and a stage error code (*e.g.* `E08_HYP_INVALID`). The executor checks that inputs exist and outputs are non-empty before advancing.

Lesson overlay. The five highest-ranked lessons (§3.5) are prepended to the prompt of each LLM stage except 8, 12–14, and 23, whose prompts are assembled by their own sub-prompt modules.

B PROMPT DESIGN

B.1 PROMPT ARCHITECTURE

The prompt system is organized into three nested layers.

Primary layer. The layer has twenty stage-level prompt templates (stages 12, 13, and 23 have no stage template and combine code with sub-prompts such as code repair and relevance scoring); each comprises a system message, a structured user template, an optional `json_mode` flag, and a `max_tokens` override. Section-level word-count targets are enforced post-generation: abstract

Algorithm 1 AUTORESEARCHCLAW control flow with domain routing

Require: Research idea $\mathcal{I}$, lesson store $\mathcal{L}$, decision budget N_d=2 shared by REFINE and PIVOT
Ensure: Paper $\mathcal{P}$ with number- and citation-check reports, updated lesson store $\mathcal{L}'$

Detect hardware profile; detect research domain D via a cascade (forced override $\rightarrow$ keyword matching $\rightarrow$ generic fallback)
Select domain prompt bank $\mathcal{B}(D)$; instantiate PROMPTMANAGER(D) and domain adapter $\mathcal{A}(D)$; load the persistent lesson store for the current research sequence, if available.
Scope topic from $\mathcal{I}$ using $\mathcal{B}(D)$.TOPIC_INIT; retrieve & screen literature; synthesize knowledge cards
$n_d \leftarrow 0$
repeat
  [Debate] K=3 domain-specialized agents via $\mathcal{A}(D)$.`debate_roles_hypothesis` $\rightarrow$ hypotheses $\mathcal{H}$ (e.g. Innovator/Pragmatist/Contrarian for ML; Theorist/Phenomenologist/Experimentalist for HEP-ph)
  Design experiments from $\mathcal{H}$ with $\mathcal{A}(D)$-injected condition terminology, paradigm constraints, and compute budget
  Generate code with the blueprint-based code agent (external coding agent optional)
  Validate code: AST checks, import verification, ablation-identity detection, alignment check
  Execute in the sandbox (subprocess, or Docker with staged network policy) with the immutable harness, budget guard, domain-specific image and pre-cached datasets
  while iterative repair: fewer than 10 iterations **and** metrics still improving **and** time cap not reached **do**
    Revise code from the last run; failure diagnosis and repair; re-execute; keep the best version
  end while
  [Debate] K=3 agents via $\mathcal{A}(D)$.`debate_roles_analysis` $\rightarrow$ result analysis
  Decide $d \in \{\text{PROCEED}, \text{REFINE}, \text{PIVOT}\}$; **if** $d \neq$ PROCEED **then** $n_d \leftarrow n_d + 1$
until $d =$ PROCEED **or** $n_d \geq N_d$ (PIVOT repeats from the debate; REFINE returns to iterative repair with $\mathcal{H}$ kept)
Build measurement registry $\mathcal{R}$ from experiment results
Draft paper with verified tables from $\mathcal{R}$ using $\mathcal{A}(D)$.`preferred_template`; review $\rightarrow$ revise (length guard)
Verify: replace unmatched numbers in results sections and tables with placeholders; flag the draft
Four-layer citation verification via CrossRef, OpenAlex, arXiv, and Semantic Scholar; remove unresolved entries and entries judged irrelevant
Extract lessons; update $\mathcal{L} \rightarrow \mathcal{L}'$ with time-decayed weights
return $\mathcal{P}$ with verification reports

150–200, introduction 800–1000, related work 600–800, method 1000–1500, experiments 800–1200, results 600–800, discussion 400–600, conclusion 200–300.

Reusable block layer. Shared text fragments are inserted into several stages to standardize writing and reporting instructions: `academic_style_guide`, `anti_hedging_rules`, `writing_structure`, `dataset_guidance`, and `hp_reporting`.

Sub-prompt layer. Sub-prompts handle operations within a stage: `architecture_planning`, `generate_single_file`, `code_repair`, `hypothesis_synthesize`, `analysis_synthesize`, and `iterative_improve`.

Users can override any prompt via `prompts.default.yaml`; the PROMPTMANAGER substitutes `{word_chars}` placeholders without altering JSON schema braces.

Domain-aware prompt selection. The PROMPTMANAGER accepts a `domain` argument at instantiation and selects the corresponding stage bank. The pipeline selects one of two banks: an ML bank (default) and a HEP-ph phenomenology bank. All other domains receive the ML bank, augmented with a domain-adapter overlay where an adapter exists. Both banks expose the same 20 stage keys and the same `{placeholder}` variables, checked by a parity test suite, so downstream pipeline logic does not depend on the bank.

Domain detection runs a four-level cascade at pipeline start. *Level 0*: a forced override via `project.profile` in `config.yaml` skips all detection and ensures cross-stage consistency. *Level 1*: fast keyword matching against 32 domain rules (275 keywords) ordered most-specific-first. *Level 2*: an LLM classifier for topics not resolved by keywords (available in the detector API). *Level 3*: generic `_generic.yaml` fallback.

Beyond the two banks, AUTORESEARCHCLAW supports further domains through a profile-and-adapter system in `researchclaw/domains/`. Each domain profile (a YAML file) specifies the experiment paradigm, condition terminology, standard baselines, typical file structure, core libraries, Docker image, metric types, statistical tests, output formats, and prompt guidance blocks. Each domain adapter injects domain-specific content into stage prompts via `{domain_context}` and related placeholders. For HEP-ph, metabolic modeling, and statistics, the profile also routes execution to a domain agent: ColliderAgent (FeynRules, MadGraph5_aMC@NLO, Delphes, MadAnalysis5), a COBRApy agent for FBA, pFBA, and FVA, and a statistics agent. Quantum topics run in the native sandbox with a Qiskit skill at stages 10 and 13. Table 7 lists the configured domains.

Table 7: Domain support matrix. *Bank*: stage prompt bank (20 stage templates each). *Profiles*: YAML profiles with experiment-paradigm configuration. *Adapter*: Python adapter with stage-level prompt injection. *Agent*: domain agent that executes experiments. *Template*: preferred LaTeX template for export.

Domain	Bank	Profiles	Adapter	Agent	Template
Machine learning	ML	8	✓	–	NeurIPS
High-energy physics	HEP-ph	1	✓	collider	JHEP
Biology (incl. metabolic)	ML	4	✓	metabolic modeling	default
Statistics	ML	1	generic	statistics	generic
Quantum computing	ML	–	–	– (Qiskit skill)	default
Chemistry	ML	2	✓	–	default
Computational physics	ML	3	✓	–	default
Economics	ML	1	✓	–	default
Mathematics	ML	2	✓	–	default
Neuroscience	ML	2	✓	–	default
Robotics, security	ML	2	✓	–	default

Domain-differentiated debate roles. The two banks also differ in their debate personas. For hypothesis generation (Stage 8), the ML bank uses *Innovator* (cross-domain analogies, high-risk hypotheses), *Pragmatist* (computational feasibility, incremental gains), and *Contrarian* (challenges assumptions, finds failure modes). The HEP-ph bank replaces these with *Theorist* (BSM Lagrangians, symmetries, UV completions), *Phenomenologist* (testable observable signatures, analytical cross-section estimates), and *Experimentalist* (detector acceptances, background shapes, overlooked systematics). For result analysis (Stage 14), ML uses Optimist / Skeptic / Methodologist; HEP-ph uses Model-Builder / Phenomenologist / Experimentalist, with all verdicts stated in natural units with explicit experimental bound citations.

B.2 PER-STAGE PROMPT BREAKDOWN

Table 8 lists the key prompt requirements and output format of each stage in the ML bank. The ML stage bank is approximately 11K tokens, and shared blocks and sub-prompts add about 13K tokens. Two stages are detailed below: stage 1, which illustrates the shared prompt structure, and stage 10, which has the longest specification.

Stage 1 prompt (TOPIC_INIT)

Stage 1 illustrates the prompt structure shared across all stages. The *system* message defines the agent role and novelty principles:

> *"You are a rigorous research planner who identifies NOVEL, TIMELY research angles. You follow recent trends from top venues in the relevant domain and propose research that advances the frontier rather than repeating known results. NOVELTY PRINCIPLES: A good research angle addresses a GAP not yet covered by existing work. Avoid pure benchmark/comparison studies unless the methodology is novel. The research must be FEASIBLE with limited compute (single GPU, hours not days). Check: would a reviewer say 'this is already well-known'? If so, find a sharper angle."*

Table 8: ML prompt bank: key requirements and output format per stage.

#	Stage	Key requirements	Output
Phase A: Research Scoping			
1	TOPIC_INIT	Novel angle, SMART goal, trend validation, benchmark specification	Markdown
2	PROBLEM_DECOMPOSE	$\geq$4 prioritized sub-questions, risks	Markdown
Phase B: Literature Discovery			
3	SEARCH_STRATEGY	Merged search plan, source manifest	JSON
4	LITERATURE_COLLECT	$\geq$8 candidate papers with abstract, year, source	JSON
5	LITERATURE_SCREEN	Domain-match filter, recency preference, quality floor	JSON
6	KNOWLEDGE_EXTRACT	Structured knowledge cards: problem, method, data, metrics, findings	JSON
Phase C: Knowledge Synthesis & Hypothesis Debate			
7	SYNTHESIS	Cluster overview, gap list, prioritized opportunities	Markdown
8	HYPOTHESIS_GEN	$\geq$2 falsifiable hypotheses; novelty argument, measurable prediction, failure condition, required baselines	Markdown
Phase D: Experiment Design			
9	EXPERIMENT_DESIGN	YAML plan: objectives, datasets, baselines, proposed methods, ablations, metrics, risks, compute budget	YAML
10	CODE_GENERATION	Multi-file Python; algorithm integrity; calibration loop; PyTorch detach rules; condition breadth-first ordering	Multi-file code
11	RESOURCE_PLANNING	GPU/CPU time estimates, parallelism plan, fallback strategy	JSON
Phase E: Self-Healing Execution			
12	EXPERIMENT_RUN	Execution stage; no LLM call	*metrics.json*
13	ITERATIVE_REFINE	Revise code from the previous run's output; failure diagnosis and repair (iterative repair)	Patched code
Phase F: Analysis & Decision			
14	RESULT_ANALYSIS	Per-hypothesis verdict; effect sizes; methodology audit; quality rating 1–10; K=3 debate consensus	Markdown
15	RESEARCH_DECISION	PROCEED / REFINE / PIVOT with evidence-based justification	Markdown
Phase G: Paper Writing			
16	PAPER_OUTLINE	Section structure, 3 title candidates, per-section content plan	Markdown
17	PAPER_DRAFT	Full 8–10 page draft; verified-registry tables only; no fabricated numbers in strict sections	Markdown
18	PEER_REVIEW	$\geq$3 reviewer perspectives; soundness, novelty, presentation, significance	Markdown
19	PAPER_REVISION	Point-by-point response to reviews; length guard	Markdown
Phase H: Finalization & Verification			
20	QUALITY_GATE	Quality report; noncritical warnings do not block	JSON
21	KNOWLEDGE_ARCHIVE	Lesson extraction; time-decayed update	Markdown
22	EXPORT_PUBLISH	LaTeX + sanitized BibTeX; domain-template selection; numeric verification pass	LaTeX
23	CITATION_VERIFY	Four-layer citation verification (CrossRef, OpenAlex, arXiv, S2); remove unresolved and irrelevant entries	JSON

The *user* template is parameterised by `{topic}`, `{domains}`, `{project_name}`, and `{quality_threshold}`, and requires six output sections: **Topic**, **Novel Angle** (specific unexplored aspect with trend validation, with no fabricated paper titles), **Scope**, **SMART Goal**, **Constraints**, and **Success Criteria**.

Stage 10 prompt (CODE_GENERATION)

Stage 10 carries the highest prompt-engineering load (about 350 lines, or 4.5K tokens, of specification). The prompt states four requirements.

Algorithm integrity. A method labeled "Bayesian Optimization" must include a surrogate model, acquisition function, and model updates. A method labeled "PPO" must include a clipped surrogate objective, learned value baseline, and `clip_eps` in the loss. Dead parameters (declared but never consumed) are forbidden.

Pilot-run check. The prompt asks the generated code to run a pilot (3–5 seeds, 2 conditions) and check that the primary metric varies across conditions. If all conditions score identically, the generated code is instructed to print `WARNING: DEGENERATE_METRICS` and adjust task difficulty; the pipeline logs the warning (§3.2).

PyTorch detach rules. Any tensor from a previous forward pass used in the current loss must be `.detach()`'d. This prevents the "backward through the graph a second time" error.

Condition breadth-first ordering. One repetition per condition executes before any parameter sweep begins, so an interrupted run still has results for every condition.

C EXECUTION AND SECURITY MODEL

ARC supports two execution backends. All experiments in this paper use the *subprocess sandbox*, which runs generated code as a local process under a wall-clock budget after static screening. The optional *Docker backend* runs each experiment in a dedicated container that is removed after execution (`--rm`), runs as the host user's UID:GID, not root, and applies resource limits (8 GB memory, 2 GB shared memory, configurable wall-clock timeout). The image pre-installs a scientific Python stack (PyTorch, torchvision, transformers, datasets, gymnasium, SciPy, scikit-learn, pandas, networkx, and others), and common datasets (CIFAR-10, CIFAR-100, FashionMNIST) are mounted read-only.

Staged network policy (Docker backend). The default `setup_only` policy enables network access during dependency installation and data acquisition and disables it with `iptables` rules before the experiment script starts; the rules require the container to hold the corresponding capability, and without it the backend logs a warning and the run keeps the container's default network access. Alternative policies are `none`, `full`, and `pip_only`.

Code validation. (1) AST parsing for syntactic correctness; (2) AST security checks for forbidden calls (`os.system`, `os.popen`, `subprocess.run`, `shutil.rmtree`) and banned builtins (`eval`, `exec`, `compile`, `__import__`); (3) a module blacklist (`subprocess`, `socket`, `http`, `urllib`, `requests`, `ftplib`, `smtplib`, `ctypes`, `signal`); (4) import validation against an allowlist. Violations are classified as errors (block) or warnings (log). The immutable harness (`report_metric()`, `finalize()`, `should_stop()`) standardizes metric output; it is copied into each run directory before the generated files, and a generated file of the same name is not copied. Metrics are parsed from standard output, and the harness standardizes their reporting.

Budget guard. Each experiment run is stopped at its wall-clock budget, the harness's `should_stop()` returns true at 80% of that budget, and iterative repair starts no new iteration after 1.5 times the budget by default. The budget guard limits time only.

Optional components. *External coding agent*: a keyword-based heuristic scores each experiment plan on six signals (component count, file-structure hints, domain-complexity terms, number of conditions, historical failures, and dependency depth) into $c \in [0, 1]$; plans with c above a configurable threshold (default 0.2) can be delegated to an external coding agent. *SmartPause*: a fixed weighted

combination of stage criticality, the within-run rejection rate, and a process-reward score (when available) pauses the pipeline when it falls below 0.7; without a process-reward model it does not trigger. *Docker backend*: described above. None of the three is enabled in the experiments reported in this paper.

D ARC-BENCH DETAILS

D.1 BENCHMARK ARCHITECTURE

Topic specification. Each topic is a YAML entry with an identifier, a topic statement naming the compared methods, domain tags, a primary metric key, and its optimization direction; each domain additionally provides rubric files and, where needed, a toolchain profile. ML topics satisfy three criteria: (1) CPU-executable in under ten minutes on a single core with `numpy`/`scipy`/`sklearn`-level dependencies; (2) a genuine scientific comparison between at least two algorithmic approaches; (3) structured quantitative outputs that a judge can evaluate from artifacts. ML01–ML10 are also used in the HITL ablation and the component ablation. Table 9 lists the ML topics; physics, biology, statistics, and quantum topics are listed in Table 10.

Table 9: ARC-BENCH ML topics and their primary metrics.

ID	Comparison	Metric
ML01	Dropout variants (standard, spatial, MC/variational) vs. no dropout in shallow tabular MLPs; calibration and accuracy	test accuracy
ML02	Ensembles (bagging, boosting, stacking) for noisy non-linear regression	RMSE
ML03	Gradient-free optimizers (Nelder–Mead, Powell, CMA-ES) on non-convex functions	best objective value
ML04	Feature scaling (standard, min-max, robust) for k-NN classification	test accuracy
ML05	Dimensionality reduction (PCA, t-SNE, UMAP) for cluster preservation	silhouette
ML06	Learning-rate schedules (cosine, step, exponential, warm restarts) for logistic regression	epochs to converge
ML07	Text features (TF-IDF, counts, hashing) with Naive Bayes and SVM on 20 Newsgroups	macro-F1
ML08	Class-imbalance handling (SMOTE, oversampling, class weights, undersampling)	balanced accuracy
ML09	Bayesian optimization vs. grid vs. random search for random forests	best CV score
ML10	Cross-validation strategies (k-fold, stratified, repeated, LOO) at small sample sizes	estimation bias
ML11	Outlier detectors (Isolation Forest, LOF, one-class SVM, elliptic envelope)	ROC-AUC
ML12	Clustering (k-means, agglomerative, DBSCAN, spectral) on synthetic shapes	ARI
ML13	GP kernels (RBF, polynomial, Matérn-3/2, Matérn-5/2) on 1-D and 5-D functions	test NLL
ML14	Conformal prediction (split, Mondrian, CQR) under heteroscedastic noise	coverage gap
ML15	Filter feature selection (MI, χ^2, F-test) vs. L1-logistic selection	test accuracy
ML16	Bandits (ϵ-greedy, UCB1, Thompson sampling, Exp3) under stationary and drifting arms	cumulative regret
ML17	Topic models (LDA, NMF, LSA) on a 20 Newsgroups subset	c_v coherence
ML18	Probability calibration (Platt, isotonic, temperature) for RF, GBM, and SVM	ECE
ML19	Semi-supervised learning (label propagation, label spreading, self-training)	test accuracy
ML20	Classical forecasters (AR, ARIMA, SARIMAX, ETS, Theta) on synthetic seasonal series	sMAPE
ML21	Causal discovery (PC, GES, NOTEARS-linear) on synthetic linear SEMs	SHD
ML22	Active learning (uncertainty, margin, query-by-committee, expected error reduction)	accuracy at budget
ML23	Learning to rank (pointwise, pairwise RankNet-lite, listwise ListMLE-lite)	NDCG@10
ML24	Online learning (SGD, passive-aggressive, online NB, FTRL) under concept drift	prequential accuracy
ML25	Echo state network vs. matched MLP vs. GP for Lorenz-63 prediction	valid prediction time

Rubric structure. Each topic ships with a rubric file defining a hierarchical tree of 8–11 leaf criteria across three categories:

Table 10: ARC-BENCH topics outside machine learning.

ID	Topic
P01	Warped-extra-dimension Kaluza–Klein graviton resonance (hep-ph/9909255)
P02	Heavy Majorana neutrino production $pp \to \mu^{\pm} N$ at 7/8/14 TeV (1308.2209)
P03–P04	B–L Z' with kinetic mixing: 8 TeV dilepton exclusion and 13 TeV cross-section slices (1605.02910)
P05	Photophobic ALP E_T^{miss} shape in $pp \to aW^{\pm}\gamma$ (1701.05379)
P06	U_1 vector leptoquark exclusion from an ATLAS+CMS $pp \to \tau\nu$ recast (1811.07920)
P07	Scalar leptoquark m_{ej} resonance via the LUXlep lepton PDF (2005.06475)
P08	SSM and E_6-ψ Z' dimuon cross sections at 13 TeV (2103.02708)
P09–P10	U_1 leptoquark shapes and exclusion/discovery contours at muon colliders (2104.05720)
B01–B03	*E. coli* succinate and lactate knockout screens; acetate overflow phase map
B04–B06	*S. cerevisiae* ethanol yield; *M. tuberculosis* essentiality and drug targets; *E. coli* carbon-source robustness
B07	Reproducible FBA protocol benchmark: FBA vs. pFBA vs. loopless FBA/FVA
S01	Bootstrap confidence-interval coverage under light-tailed, skewed, and heavy-tailed data
S02	Double machine learning for average treatment effects under nonlinear confounding
S03	Reliability of LLM-assisted statistical model selection under assumption violations
Q01–Q02	Data encodings for variational quantum classifiers; ablating entangling gates
Q03–Q05	VQE optimizers under shot noise; data re-uploading expressivity; barren plateaus and cost locality
Q06–Q08	Neural warm starts for QAOA; MPS classifiers vs. neural networks; layerwise training
Q09–Q10	Noise-aware VQC training; quantum autoencoders for state compression

- **Code Development (CD)**, weight 25: correctness and completeness of algorithm implementation.
- **Code Execution (CE)**, weight 25: whether the code ran successfully and produced machine-readable metric artifacts with multi-seed dispersion reporting.
- **Result Analysis (RA)**, weight 50: quality of the scientific write-up: per-hypothesis verdicts supported by reported numbers, appropriate caveats, and no unsupported values.

Each leaf carries a weight, the leaf weights within a category sum to that category's weight (25, 25, or 50), and each leaf receives a score in $[0, 1]$. Aggregate scores are:

$$\texttt{overall_strict} = \textstyle\sum_{\ell} w_\ell \cdot s_\ell \;/\; 100 \tag{2}$$

$$\texttt{results_only} = \textstyle\sum_{\ell \in \text{CE} \cup \text{RA}} w_\ell \cdot s_\ell \;/\; 75 \tag{3}$$

All main-text comparisons use `overall_strict`; Table 14 reports alternative aggregations. The complete rubric for ML01 is shown in the box at the end of Appendix D.2.

Judge system. Each (system $\times$ topic) cell is graded against five artifact sources: (1) agent-produced code; (2) execution artifacts (CSVs, JSON metric files, stdout logs); (3) the agent-written README and claims file; (4) the topic rubric; (5) the topic manifest defining the research question, conditions, metrics, datasets, and hypotheses. The judge outputs per-leaf scores, rationale strings, and the two aggregates in Equations 2–3.

D.2 STRICT JUDGE PROTOCOL

The strict judge prompt and its JSON output schema are used by a Claude Code agent (Claude Opus 4.7) and a Codex CLI agent. Grading runs in a separate environment on de-identified copies of the run directories, with system names removed. Artifact sources and rubric structure follow Appendix D.1.

Strict criteria. Four criteria apply uniformly to every leaf. (1) **Implementation correctness**: the algorithm is verified by reading code, not by label. Common failure patterns include MC-dropout that disables dropout at eval time, CMA-ES without covariance update, and NOTEARS implemented as plain Lasso. (2) **Number grounding**: every numerical claim in the writeup must trace to a captured artifact; unsupported numbers reduce the relevant leaf to 0.1–0.3. (3) **Verdict-data consistency**: a

claimed supported hypothesis must be backed by measured evidence in the same direction; inverted verdicts score 0.1–0.3 regardless of prose quality. (4) **Coverage**: missing conditions, datasets, or seeds reduce execution leaves proportionally to manifest coverage.

Timeout rule. If a run exceeds its wall-clock budget, CE leaves are set to 0; CD and RA leaves are graded on the code and write-up produced before the timeout. This rule applied to six AIDE runs in the ML comparison. In the non-ML runs of Appendix D.3, the AIDE runs that exhausted the budget or crashed produced no write-up and scored 0.

Two passes. The two agents grade each cell independently. Per-leaf scores with $|\Delta| > 0.20$ are flagged and re-adjudicated, and final scores are the mean of the two passes.

Table 11: Rubric for ML01 (dropout variants for shallow MLPs on tabular data), abbreviated from the box below. Leaf weights sum to the category weight.

Cat.	Weight	Requirement
CD	10	No-dropout baseline, standard dropout, and an MC-style variant as distinct code paths in a shared MLP
CD	5	A shallow MLP trained with a supervised classification objective
CD	5	Several tabular datasets with a train/test split
CD	5	MC dropout averages several stochastic forward passes at test time
CE	16.7	A machine-readable metrics file with accuracy and a calibration metric (ECE, NLL, or Brier) for all conditions on at least one dataset
CE	8.3	Metrics aggregated over several seeds per condition and dataset, with dispersion
RA	20	H1: calibration of MC-style vs. standard dropout, with a verdict (supported, refuted, or inconclusive)
RA	10	H2: whether accuracy differences among variants are small relative to calibration differences
RA	10	H3: calibration of the no-dropout baseline vs. at least one dropout variant
RA	10	Write-up reports setup, key numbers, per-hypothesis outcomes, and caveats on seeds and scope

The box below shows the full rubric for ML01.

ARC-BENCH sample rubric: ML01 (dropout regularization and calibration)

Research question. Comparing dropout regularization strategies (standard, spatial, variational) for preventing overfitting in shallow MLPs on tabular classification benchmarks. Does dropout-variant choice affect calibration more than accuracy?

Code Development (CD), weight 25

`t01-code-variants` [10.0] Implement the main dropout variants: no-dropout baseline, standard element-wise dropout at one or more rates, and at least one stochastic-test-time / MC-style variant, as *distinct code paths* within a shared MLP architecture.

`t01-code-mlp` [5.0] Shallow tabular MLP (small number of hidden layers, modest width) trained with a standard supervised classification objective on sklearn benchmarks.

`t01-code-datasets` [5.0] Load multiple tabular sklearn datasets (*e.g.* `breast_cancer`, `wine`, `digits`) with a reasonable train/test split.

`t01-code-mc-pass` [5.0] MC-dropout performs *multiple* stochastic forward passes at test time and averages predicted probabilities; a single deterministic pass does not qualify.

Code Execution (CE), weight 25

`t01-exec-metrics` [16.67] Machine-readable artifact (`results/metrics.json` or `stage-14/experiment_summary.json`) with numeric accuracy *and* a calibration metric (ECE, NLL, or Brier score) covering all conditions on at least one dataset.

`t01-exec-seeds` [8.33] Metrics aggregated over multiple random seeds per (condition, dataset) cell with dispersion reporting: std, stderr, CI, or min/max across seeds. A small-but-honest seed count with reported variance is preferable to a single deterministic run.

Result Analysis (RA), weight 50

`t01-result-h1` [20.0] Compare calibration (ECE or analogous) between MC-style and standard dropout across evaluated datasets; convey whether MC calibration tends to be better (H1 supported / refuted / inconclusive).

`t01-result-h2` [10.0] Discuss whether accuracy differences among variants are small, *i.e.* calibration is the discriminative axis, not accuracy. Qualitative comparison grounded in reported numbers suffices; exact percentage-point thresholds are not required.

`t01-result-h3` [10.0] Compare no-dropout baseline vs. at least one dropout variant on calibration; convey whether baseline is clearly worse, comparable, or better.

`t01-result-writeup` [10.0] README or writeup describes setup, presents key accuracy and calibration numbers, and conveys per-hypothesis outcomes (supported / refuted / inconclusive) with appropriate caveats on seed count, dataset scope, or calibration-metric choice.

Judging note. Score on scientific substance and directional correctness of evidence, not on exact threshold satisfaction. Partial but well-motivated evidence deserves partial credit; rigid wording or name mismatches should not penalize a substantively correct experiment. Aggregate: `overall` $= \sum_\ell w_\ell s_\ell / 100$; `results_only` $= \sum_{\ell \in \mathrm{CE} \cup \mathrm{RA}} w_\ell s_\ell / 75$.

D.3 Study Configurations

Baselines. In the ML comparison, AIDE and AI Scientist v2 run from their released code with the same topic manifest (question, conditions, metrics, datasets, hypotheses) as ARC, and all three systems use GPT-5.3-codex for every LLM role. All three systems run under the same total wall-clock budget of 1,800 s per run; a run that exceeds it receives CE $= 0$ (six AIDE runs). ARC runs stages 10–14 with the manifest's hypotheses and plan in place of stages 7–9, and the write-up of each system is graded as produced.

Cross-run evolution. The cross-run study separates replication at the first memory-enabled iteration from longitudinal accumulation across later iterations. Three independent fresh runs reset the lesson store and provide the no-memory reference. We separately execute three independent memory-enabled Iteration-1 runs. One of these is the first point of the persistent trajectory in Table 3; that sequence is subsequently continued for Iterations 2 and 3 with lessons retained between cycles. Run indices below denote independent repetitions and are not treated as paired trials.

Table 12: Independent replication of the first memory-enabled iteration on ML01–ML25. The three fresh runs reset the lesson store; the three memory-enabled runs are independent executions at Iteration 1. Δ is the difference between the two three-run means.

	Fresh / No Memory				Memory-Enabled Iteration 1				
Metric	**Run 1**	**Run 2**	**Run 3**	**Mean**	**Run 1**	**Run 2**	**Run 3**	**Mean**	**Δ**
CD	0.938	0.844	0.912	0.898	0.948	0.916	0.962	0.942	+0.044
CE	0.562	0.426	0.534	0.507	0.574	0.562	0.548	0.561	+0.054
RA	0.442	0.358	0.476	0.425	0.612	0.568	0.584	0.588	+0.163
Overall	**0.596**	**0.497**	**0.600**	**0.564**	**0.687**	**0.654**	**0.670**	**0.670**	**+0.106**

E Per-Topic Scores and Sensitivity

Per-topic scores. Table 13 reports the category and Overall scores for every topic. Paired differences in Table 2, Figure 3, and the sensitivity analyses of §4.2 are computed from these values; Wilcoxon tests use unrounded scores.

Category decomposition. Against AI Scientist v2, all three categories contribute to the weighted margin: CD by 0.056, CE by 0.030, and RA by 0.091, with ARC higher on CD on 23 of 25 topics. Against AIDE, RA contributes 0.053, CE 0.037, and CD -0.005. RA is correlated with CE across topics (r=0.67 for ARC), because an analysis can only cite runs that finished; on the six topics where both ARC and AIDE reach CE ≥ 0.5, ARC's RA is 0.18 higher on average.

Table 13: Per-topic category and Overall scores on ML01–ML25 (CD:CE:RA = 25:25:50). Displayed category scores are rounded to two decimals, while Overall scores and statistical analyses are computed from the underlying unrounded leaf scores. *AIDE run exhausted the budget (CE set to 0). ASv2: AI Scientist v2.

	ARC (Full-Auto)				AIDE				ASv2			
Topic	CD	CE	RA	Overall	CD	CE	RA	Overall	CD	CE	RA	Overall
ML01	0.52	0.27	0.22	0.307	0.90	0.07	0.12	0.302	0.94	0.40	0.54	0.605
ML02	1.00	1.00	0.52	0.760	1.00	0.73	0.64	0.753	0.50	0.40	0.28	0.365
ML03	0.75	0.00	0.40	0.388	1.00	1.00	0.36	0.680	0.50	0.67	0.00	0.292
ML04	0.96	0.95	0.97	0.963	0.96	0.92	0.36	0.650	0.80	0.53	0.64	0.653
ML05	1.00	0.67	0.40	0.617	1.00	0.00*	0.26	0.380	0.50	0.33	0.00	0.208
ML06	1.00	0.53	0.42	0.591	0.94	0.07	0.12	0.311	0.82	0.40	0.28	0.446
ML07	1.00	0.90	0.95	0.950	0.99	0.75	0.85	0.859	0.74	0.50	0.21	0.414
ML08	1.00	0.47	0.33	0.529	1.00	0.77	0.56	0.723	0.40	0.40	0.41	0.406
ML09	1.00	0.93	0.40	0.681	0.94	0.09	0.16	0.336	0.82	0.65	0.22	0.479
ML10	0.95	0.02	0.12	0.304	1.00	0.00*	0.21	0.356	0.85	0.57	0.24	0.473
ML11	0.96	0.96	0.95	0.956	1.00	0.97	0.59	0.787	0.85	0.42	0.17	0.406
ML12	0.91	0.71	0.41	0.613	0.96	0.42	0.36	0.528	0.99	0.86	0.45	0.688
ML13	0.97	0.37	0.36	0.515	0.93	0.90	0.56	0.736	0.79	0.30	0.42	0.482
ML14	1.00	0.38	0.32	0.504	0.90	0.38	0.34	0.489	0.69	0.29	0.16	0.324
ML15	1.00	0.53	0.50	0.633	1.00	0.00*	0.26	0.380	0.79	0.38	0.25	0.418
ML16	1.00	0.70	0.45	0.650	0.97	0.07	0.18	0.350	0.79	0.55	0.44	0.554
ML17	0.96	0.93	0.44	0.694	0.96	0.00*	0.28	0.381	0.89	0.45	0.19	0.429
ML18	0.74	0.52	0.32	0.473	0.97	0.68	0.52	0.675	0.61	0.33	0.27	0.371
ML19	1.00	0.43	0.60	0.658	1.00	0.92	0.53	0.744	0.74	0.23	0.15	0.318
ML20	0.91	0.45	0.44	0.561	0.90	0.82	0.37	0.616	0.71	0.34	0.13	0.328
ML21	0.88	0.53	0.33	0.512	0.70	0.06	0.12	0.251	0.30	0.16	0.12	0.178
ML22	1.00	0.30	0.25	0.450	1.00	0.00*	0.18	0.340	0.57	0.57	0.18	0.378
ML23	1.00	0.57	0.44	0.614	0.97	0.68	0.12	0.472	0.93	0.47	0.23	0.465
ML24	1.00	0.38	0.16	0.424	0.96	0.07	0.12	0.319	0.75	0.64	0.34	0.517
ML25	0.93	0.57	0.36	0.554	1.00	0.00*	0.23	0.365	0.54	0.18	0.20	0.280
Mean	0.938	0.562	0.442	**0.596**	0.958	0.415	0.336	**0.511**	0.712	0.442	0.261	**0.419**

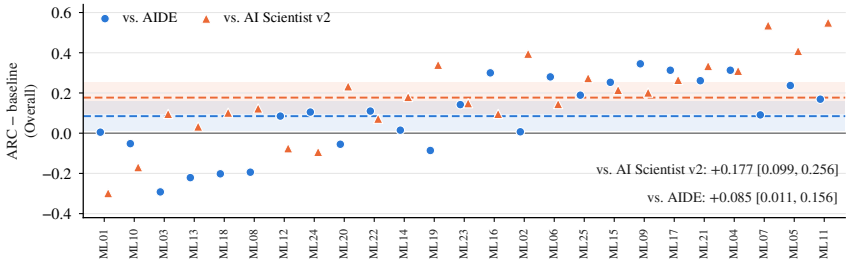

Figure 3: Per-topic difference in Overall between ARC (Full-Auto) and each baseline on ML01–ML25, sorted by the mean of the two differences. Dashed lines show the mean paired difference and shaded bands its 95% bootstrap interval (20,000 resamples). AIDE exhausted the budget on ML05, ML10, ML15, ML17, ML22, and ML25.

Rubric weights. Table 14 reports the Overall score under three alternative aggregations of the same leaf scores.

F HITL Ablation Details

Topics. The study uses ML01–ML10 (Table 9): dropout variants, ensembles, gradient-free optimization, feature scaling, dimensionality reduction, learning-rate schedules, text classification, class imbalance, hyperparameter search, and cross-validation.

Table 14: Overall score on ML01–ML25 under alternative aggregations of the same leaf scores. The ranking of the systems is the same under all four.

Aggregation	ARC (Full-Auto)	AIDE	AI Scientist v2
CD:CE:RA = 25:25:50 (Table 2)	0.596	0.511	0.419
Leaf weights before rescaling	0.618	0.536	0.442
Equal category weights	0.647	0.570	0.472
Results only (CE and RA)	0.482	0.362	0.321

Pauses and payloads. Table 16 lists the stages at which each regime pauses and the payloads it receives. For each topic, there is one payload for each of stages 5, 8, 9, 14, 17, and 20 (60 payloads). Each payload was drafted by an LLM from the topic manifest and preliminary Full-Auto runs and then reviewed and edited by the authors. Preparing the guidance in advance keeps its content identical across regimes and reproducible from the released files; guidance typed live at each pause would vary with the researcher and with what earlier regimes had shown. Pre- and Post-Experiment payload files are identical in content to the corresponding stages of the six-stage file, and all 60 payloads are released as ten JSON files. A scripted adapter answers every pause: it returns the payload as written guidance when one exists for that stage and approves the stage output otherwise.

Scoring. The stage-20 score is the `score_1_to_10` field of the quality report produced by the pipeline's LLM quality gate (GPT-5.3-codex), which scores each manuscript in two independent passes, each in a fresh context, that are averaged, with the same rubric for every regime and configuration. In regimes that pause at stage 20, the payload is delivered after the gate has produced its score, and the gate prompt does not read intervention files. Each (topic, regime) pair has one scored run.

Human check. Three reviewers not involved in developing ARC applied the stage-20 rubric to every completed manuscript of the HITL ablation and the component ablation and recorded pass or fail without the gate's score. Table 15 lists the counts per reviewer, and Tables 4 and 5 report their means. For the registry row of Table 5, the reviewers reported the numbers absent from the measurement records and did not record a pass decision; that count is the *Unsupp.* column.

Table 15: Human pass counts per reviewer (manuscripts passed, out of ten).

Study	Row	R1	R2	R3	Mean
HITL	Full-Auto	2	3	2	2.3
HITL	Gate-Only	4	4	4	4.0
HITL	CoPilot	6	7	6	6.3
HITL	Thorough	4	3	3	3.3
HITL	Step-by-Step	5	6	6	5.7
HITL	Pre-Experiment	4	4	3	3.7
HITL	Post-Experiment	4	3	3	3.3
Ablation	Full ARC	2	3	2	2.3
Ablation	without debate	0	1	1	0.7
Ablation	without self-healing	0	0	0	0.0
Ablation	without registry	–	–	–	–
Ablation	without debate and self-healing	0	0	1	0.3

G Component Ablation Notes

Best-of-three selection. Each configuration–topic pair is run three times and the best-scoring completed run is kept, under the same rule for every row. *Compl.* counts topics for which at least one of the three runs produced a scored manuscript.

Debate row. The *without debate* row disables both debates (stage 8 and stage 14), replacing each with a single-agent call that produces the same artifact.

Table 16: Pause stages and delivered payloads per regime. Payload stages are 5, 8, 9, 14, 17, and 20; the stage-20 payload arrives after scoring.

Regime	Pause stages	Payloads delivered
Full-Auto	none	none
Gate-Only	5, 9, 20	5, 9, 20
Thorough	phase boundaries: 2, 6, 8, 11, 13, 15, 19, 23	8
CoPilot	1, 2, 3, 5, 7, 8, 9, 10, 13, 15, 16, 17, 18, 20, 23	5, 8, 9, 17, 20
Step-by-Step	all 23 stages	5, 8, 9, 14, 17, 20
Pre-Experiment	all 23 stages	5, 8, 9
Post-Experiment	all 23 stages	14, 17, 20

H Design-Space Exploration

These preliminary observations guided the default settings.

Number of debate agents (K). We compared $K \in \{2, 3, 5\}$ over 10 runs each; hypothesis diversity is measured over the synthesized hypotheses of each run. With K=2, hypothesis diversity was 23% lower than with K=3 and the discussion reduced to arguments for and against a single idea. With K=5, token use rose by 67% and diversity by 8% relative to K=3, and the two additional agents mostly repeated the positions of the first three. We therefore use K=3 for ML topics.

Lesson half-life ($T_{1/2}$). We compared $T_{1/2} \in \{7, 15, 30, 60, \infty\}$ days over sequences of runs that reused the lesson store. With $T_{1/2}$=7, lessons lost weight before they could affect later runs, and with $T_{1/2}$=∞, contradictory lessons accumulated after about 15 runs. $T_{1/2}$=30 gave the best results in this comparison, and we use it as the default.

I Case Study Details

Table 17 summarizes the two ML10 runs discussed in §4.6.

Table 17: Case study on ML10 (cross-validation strategies for small-sample model selection).

	Full-Auto (stage-20 score 4.0)	CoPilot (stage-20 score 8.0)
Guidance	None.	Payloads at stages 5, 8, 9, 17 (and 20, after scoring); the stage-9 payload asks for distinguishable strategies, LOOCV only for $n \leq 50$, and only logged metrics.
Design	Eight strategy configurations across four families (k-fold, stratified, repeated, LOOCV).	Nine selection pipelines: LOO, non-stratified 5-fold, stratified 3- and 5-fold, repeated stratified 5-fold and a stability-weighted variant, nested 2×2 CV with a ResNet-18, and pipelines labeled CORAL and DANN.
Execution	Every configuration reports an estimation bias of 0 with zero variance. The pilot-run check flags identical conditions, and the run continues.	Most pipelines differ in bias; two pairs are identical (non-stratified vs. stratified 5-fold; repeated stratified 5-fold vs. its stability-weighted variant).
Manuscript	Reports that all strategies perform identically and cannot rank them.	Reports the bias differences and states that only estimation bias was logged and that the results come from a single run.

J Failure Analysis

All 13 incomplete runs in Table 4 occur outside Gate-Only and Step-by-Step, and 11 of them fail at stage 17 (`paper_draft`). Stage 17 does not start drafting when no usable metric exists

upstream. The logs show four recurring causes: (a) stages 10–14 do not produce a usable metric table; (b) environment or dependency errors, such as missing `imblearn` or `LightGBM`; (c) a planned dataset (*e.g.* FashionMNIST) cannot be loaded in the sandbox; (d) the design is too large for the budget or invalid, for example an invalid cross-validation setting (`k must be in [2, 34]`), so only a small fraction of conditions complete. The stage-17 check groups these causes under a single failure code.